检测 C++ 顺序容器中元素访问错误

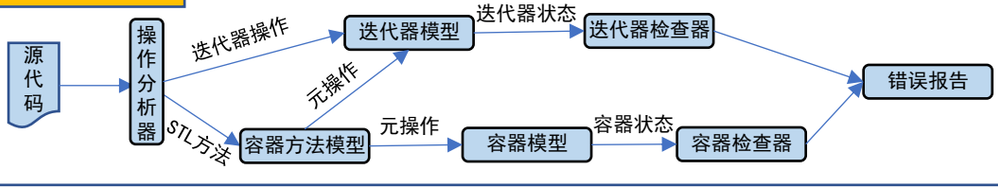
the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024: 883-894.

李知霖（lizhilin24@otcaix.iscas.ac.cn），马旭桐，胡梦泽，严俊
软件工程技术研究开发中心，基础软件与系统重点实验室

背景介绍

C++标准模板库（STL）中的顺序容器（Sequence Containers，简称 SC），例如 vector，因其良好的可维护性和灵活性，在大型软件项目中被广泛应用。这些容器提供了高效的数据存储和访问接口，使得程序设计更加模块化和易于扩展。然而，访问顺序容器中的元素时，如果没有在编译期或运行时进行边界检查，极易引发越界访问等安全漏洞。更为棘手的是，现有的大多数静态分析工具难以准确识别这类错误，原因在于它们缺乏对顺序容器的动态行为以及迭代器操作的精确模型，导致无法有效追踪容器的大小变化和迭代器所指向的具体元素。因此，准确分析和检测顺序容器元素访问的安全性，仍然是静态程序分析领域的一大挑战。

方法架构



迭代器操作的操作语义

sc：顺序容器，δ：偏移量，ϕ：有效性

Type	Expression	Operational Semantics
Assignment	Assign(it_1,it_2)	$\frac{it_1=(id,ct,\tau,\phi)}{it_1=(id,it_2.ct,it_2.\tau,it_2.\phi)}$
Movement(Increase)	Increase(it,n)	$\frac{it=(id,ct,\tau,\phi)}{it=(id,ct,\tau+n,\phi)}$
Movement(Decrease)	Decrease(it,n)	$\frac{it=(id,ct,\tau,\phi)}{it=(id,ct,\tau-n,\phi)}$
Invalidation	Invalid(InSCIt)	$\frac{InSCIt=(id,ct,\tau,\phi)}{InSCIt=(id,ct,\tau,false)}$
Bind	Bind(it,sc)	$\frac{it=(id,unknown,\tau,\phi)}{it=(id,sc,\tau,\phi)}$

元操作的操作语义

Δ：大小变化，sc：顺序容器，σ：大小，τ：类型

Type	Expression	Operational Semantics
Construct	Construct(sc,Δ)	$\frac{sc=(id,\theta,\tau)}{sc=(id,\Delta,\tau)}$
Resize	Resize(sc,Δ,InSCIt)	$\frac{sc=(id,\theta,\tau)}{sc=(id,\Delta,\tau);\ Invalid(InSCIt)}$
Add	Add(sc,Δ,InSCIt)	$\frac{sc=(id,\theta,\tau)}{Resize(sc,\theta+\Delta,InSCIt)}$
Sub	Sub(sc,Δ,InSCIt)	$\frac{sc=(id,\theta,\tau)}{Resize(sc,\theta-\Delta,InSCIt)}$
Swap	Swap(sc_1,sc_2,InSCIt)	$\frac{sc_1=(id_1,\theta_1,\tau);\ sc_2=(id_2,\theta_2,\tau)}{sc_1=(id,\theta_2,\tau);\ sc_2=(id,\theta_1,\tau);\ Invalid(InSCIt)}$

容器迭代器错误检测

```
for all n in Path do
  if Equal(n,Sub) then
    ct = GetContainer(n);
    change = GetParam(n);
    if ct.size − change < 0 then
      BugReport(Empty Container Access Bug);
  if Equal(n,Increase) then
    it = GetIterator(n);
    ct = it.ct;
    change = GetParam(n);
    if it.offset + change > ct.size then
      BugReport(Iterator Out-of-Bound Access Bug);
  if Equal(n,Decrease) then
    it = GetIterator(n);
    change = GetParam(n);
    if it.offset - change < 0 then
      BugReport(Iterator Out-of-Bound Access Bug);
  if IsAccessOperation(n) then
    it = GetIterator(n);
    ct = it.ct;
    if ct.size == 0 then
      BugReport(Empty Container Access Bug);
    if not it.valid then
      BugReport(Invalid Iterator Access Bug);
    if it.offset < 0 or it.offset >= ct.size then
      BugReport(Iterator Out-of-Bound Access Bug);
```

Path指通过静态分析获得的、由AST节点组成的执行路径

将STL函数抽象为元操作

Meta-Operation	Example
Construct	construct function
Resize	resize, assign, clear
Add	insert, emplace, push_back
Sub	erase, pop_back, pop_front
Swap	swap
Other	find, begin

SCASA 用于检测顺序容器元素访问错误的静态分析工具

- 基于经典静态分析工具Clang Static Analyser（CSA）
- 提供能够更有效地表示STL中顺序容器和迭代器的状态的联合模型
- 提供用于检测三类典型的顺序容器漏洞模式的检查器

工具github链接：
https://github.com/SQUARE-RG/Scasa

创新及贡献

- 我们对 STL 中的顺序容器及其迭代器进行了建模，并通过一组元操作（meta-operations）对 STL 方法进行了表示。这一组合模型能够更好地表示顺序容器与迭代器的状态。
- 我们开发了一种分析器Scasa，配备了专门设计的检查器，能够检测三类顺序容器的错误模式，从而系统性地识别 C++ 项目中的 EASC（错误使用顺序容器）错误。
- 借助我们对真实项目分析所生成的错误报告，我们向开发者提供了反馈，并揭示出了一些此前未被发现的缺陷。

实验评估结果

Scasa对顺序容器状态修改方法的 AST 节点的覆盖与分析情况统计

Project	Total	*Scasa*		CSA	
		Covered	Analyzed	Covered	Analyzed
Aria2	547	487	487	524	480
Assimp	2,258	1,651	1,650	1,266	1,230
Barony	1,340	930	930	830	718
Bitcoin	2,069	1,568	1,567	1,512	1,441
Cataclysm-DDA	5,134	3,734	3,733	3,513	3,265
Jsoncons	1,384	896	896	836	802
Leveldb	352	142	142	149	148
Rgbds	150	109	109	106	100
Total	13,234	9,530	9,527	8,745	8,193

人工数据集上的分析结果

Tool	*Scasa*	CSA	Cppcheck	Infer
TP	2,080	320	1,210	0
TN	0	0	0	0
FP	30	0	0	0
FN	150	1,910	1,020	2,030
Precision	98.57%	100.00%	100.00%	N/A
Recall	93.27%	14.34%	54.26%	N/A
F1	95.84%	25.08%	70.34%	N/A

真实项目上的结果

Project	*Scasa*	CSA	Cppcheck
Aria2	2/4	0/1	0/0
Assimp	17/27	6/10	0/0
Barony	3/10	0/3	0/0
Bitcoin	4/11	0/3	0/0
Cataclysm-DDA	44/70	1/30	1/4
Jsoncons	0/0	0/0	0/0
Leveldb	1/2	0/3	0/0
Rgbds	1/1	0/1	0/0
Total	72/125	7/51	1/4

分析效率：在真实项目中，Scasa 相较于 CSA 带来了 5–85% 的额外时间开销。